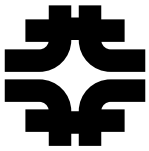

Fermi National Accelerator Laboratory

FERMILAB-Conf-96/077

Object-Oriented Geometry Engine for Monte Carlo Simulations

O.E. Krivosheev and N.V. Mokhov

Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

April 1996

Submitted to the *Radiation Protection and Shielding Topical Meeting*,
North Falmouth, Massachusetts, April 21-25, 1996

Operated by Universities Research Association Inc. under Contract No. DE-AC02-76CHO3000 with the United States Department of Energy

Disclaimer

OBJECT-ORIENTED GEOMETRY ENGINE FOR MONTE CARLO SIMULATIONS

Oleg E. Krivosheev Nikolai V. Mokhov

Fermi National Accelerator Laboratory, Batavia, IL 60510
(708) 840-4409

ABSTRACT

The object-oriented geometry engine for Monte Carlo particle transport simulations is described. The C++ class library is based on the Constructive Solid Geometry approach, which allows constructing an arbitrary geometry in terms of boolean operations on finite closed *primitive* bodies. It allows one to build hierarchical, tree-like structures which contain elementary and composite bodies. The paper covers an internal structure of the engine and its use in the frame of the MARS code system. Implementation issues and optimization techniques are discussed. The geometry engine is tied with a visualization subsystem, which is based on the Open Graphics Language (OpenGL). The subsystem handles wire-framed and solid bodies, supports orthonormal or perspective projections, rotations and shifts, clipping by arbitrary planes, lighting with different light sources. The user interface based on the *Tcl-Tk* package is also described.

1. INTRODUCTION

The use of the Monte Carlo simulation method in solving shielding, energy deposition and other particle transport problems is common nowadays. The Monte Carlo codes may include an interface to detector and accelerator component databases, drawing and *CAD* programs which handles complicated geometry in scientific and engineering applications. Exciting opportunities in this area are related to an implementation of the object-oriented (*OO*) languages and techniques.

Any simulation code can be splitted into few almost independent parts: geometry description, tracking, event generator, physics interactions etc. The well known codes (e.g., GEANT [1] and MARS [2]) developed over many years, are written in *FORTRAN*. The most promising advantages of the object-oriented approach (for geometry modeling especially) is almost direct mapping between real world description and abstract data handling possibilities, provided by the *OO* language.

2. GEOMETRY DESCRIPTION

There are two standard (along with various modifications) approaches to the geometry modeling that are used in different Monte Carlo codes: boundary surface representation and constructive solid geometry.

In the *Boundary surfaces* representation, zones with different tracking properties are built of the mathematically defined surfaces. From a practical point of view, the chosen surfaces are usually described by the low-order mathematical equations. Each zone consists of a list of surfaces with proper normal vectors in each point. This approach is rather general and, in principle, can give one a possibility to work with any surface, described by a mesh of the points. However, this approach can have some performance penalty during the tracking in the complicated geometry. For the *OO* systems, this approach was chosen by GISMO developers [4].

In the *Constructive Solid geometry* representation, zones are defined as combinations of the *elementary bodies* such as cube, cone, cylinder etc. A complex geometry description is built using the logical operations with volumes, such as *union*, *intersection* and *difference*. This approach has the following advantages: a simplicity in describing standard accelerator and detector components (pipes, boxes, cones, piramides etc), lower memory constraints, a provision for more internal optimization during the tracking. For the *OO* systems, this approach was chosen by the GEANT4 team [3].

For our implementation, the constructive solid geometry approach is chosen. A library of the elementary bodies inherited from an abstract base class *Shape* is created. These primitive *shapes* represent untransformed, i. e. non-rotated and non-translated, bodies positioned in a center of their own coordinate system. The elementary bodies provide (thanks to polymorphism) the uniform interface and define functions of their own, which calculate a distance to enter the body, a distance to leave the body and a location of a given point (inside or outside). The *composite bodies* are defined as a subclass of the shapes, with the same uniform interface. A data structure for the *composite body* representation [5], is an inverted binary tree (see Fig. 1). Leaf nodes are elementary solids, and internal nodes represent the logical operations. So, if one has N primitive solids in one composition, there are $(N-1)$ composite ones. Composite solids are represented by a pointer to leaves and transformations from a global to the local coordinate system for each leaf. The code classifies rays and points with respect to the solid and returns the classification to the user. The classification is based on the boolean algebra rules. An example of positioning is presented in Table 1.

Table I. Boolean rules for the *Where* method

Operation	Left	Right	Composite
Union	In	In	In
	In	Out	In
	In	Surface	In
	Out	In	In
	Out	Out	Out
	Out	Surface	Surface
	Surface	In	In
	Surface	Out	Surface
	Surface	Surface	Surface
Intersection	In	In	In
	In	Out	Out
	In	Surface	Surface
	Out	In	Out
	Out	Out	Out
	Out	Surface	Out
	Surface	In	Surface
	Surface	Out	Out
	Surface	Surface	Surface
Difference	In	In	Out
	In	Out	In
	In	Surface	Surface
	Out	In	Out
	Out	Out	Out
	Out	Surface	Out
	Surface	In	Out
	Surface	Out	Surface
	Surface	Surface	Surface

A second type of the objects is positioned one, containing a pointer to the parent volume and a list of *children* which are embedded into the *parent* without any cross-sections. These objects also contain a reference to the unpositioned shape (elementary or composite one) and a *parent-to-child* transformation. This approach, probably more complicated for implementation, is more intuitive and allows one to use a predefined composite object like *library*. The possible memory structure of objects is shown in Fig. 2.

3. IMPLEMENTATION ISSUES

The engine is implemented using the C++ object-oriented language under the Solaris OS-5.4. The compiler, GNU g++, promises high portability level due to its extremely portability. The

container management code is taken from the Standard Template Library (part of the upcoming C++ standard). The Tcl version 7.4 and Tk version 4.0 are taken as the most stable releases, which now are ported to the Windows and Macintosh environment.

There are two main classes on the top of the system, *GeometryServer* and *GeometryTracker*, which glue all parts together. The *GeometryServer* keeps a reference to the mother volume, a list of all the registered unpositioned solids, a list of all the registered transformations, and an interface to the object allocation and deallocation. The *GeometryTracker* performs particle tracking through the media, also keeping a cache information for the optimal tracking.

A memory management is implemented using the allocator approach, which hides a real memory interface and allows one, using specialised allocator, to have the objects stored in the persistent memory (object-oriented database).

The optimization technique is based on a box enclosure approach. Each node supports the orthonormal box in the world coordinate system, which encapsulates the original volume. This proxy boxes are build at the initialization stage. The tracking code first checks a possible intersection of the enclosure box and a ray, thus avoiding a search for all the nodes.

4. VISUALIZATION

The visualization subsystem is based on the OpenGL [6] 2-D / 3-D graphics library. OpenGL is a high-speed system-independent modern graphics library, which allows one to use a three-dimensional rendering with arbitrary lighting, colors and shadows. The OpenGL subsystem is included in all the modern UNIX and Windows system.

Each unpositioned volume builds its own representation as a wire-framed and a solid body (as display list in the OpenGL terms), including a set of different colors and lighting. The visualization subsystem handles also a support to the orthonormal or perspective projections, rotations and shifts. Up to six arbitrary clipping planes and up to eight light sources are supported as a minimum guaranteed by the OpenGL specification. The Tcl-Tk package [7] is chosen for the user interface because of its portability and simplicity. It provides the uniform and rich user interface and is supported for all the graphics devices (X11 and Windows GDI). The binding between OpenGL and Tcl-Tk is based on the OGLTK widget [8].

5. EXAMPLES

The visualization subsystem is adopted to display the MARS [2] code geometry input configuration. The main window consists of a model view which is rendered by the OpenGL server, and a control view, which is binded to Tcl-Tk. User can rotate the figure using sliders or just drugging object by a mouse. The buttons on the right side allow to perform a translation in any direction, to switch wire-frame / solid representations and to control the lighting. The CMS

detector geometry configuration is shown both in a wire-frame (see Fig. 3) and a solid body representations (Fig. 4) including lighting and clipping.

6. CONCLUSIONS

The object-oriented geometry module based on the constructive solid geometry approach is created. Modern visualization software is used for geometry representation, verification and for visual tracking. This is a part of the modified *OO* version of the MARS code.

References

[1] "GEANT, Detector Description and Simulation Tool", CERN, Geneva (1994).

[2] N. V. Mokhov, "The MARS code system Users Guide, version 13(95)", Fermilab-FN-628 (1995).

[3] "GEANT4 R&D project", Uniform Resource Locator: `http://wwwcn.cern.ch/pl/geant/geant4.html`.

[4] A. Breakstone, "GISMO Geometry Manual", University of Hawaii (1994).

[5] S. D. Roth, "Ray Casting for Modeling Solids", *Computer Graphics and Image Processing* **18**, 109 (1982).

[6] OpenGL Architecture Review Board, "OpenGL Reference Manual: The Official Reference Document for OpenGL", Release 1, Addison-Wesley, Reading, Massachusetts, (1992).

[7] J. Ousterhout, "Tcl and the Tk Toolkit", Addison-Wesley, Reading, Massachusetts, (1994).

[8] B. B. Bederson, Uniform Resource Locator: `http://www.cs.unm.edu/~bederson/graphics.html`

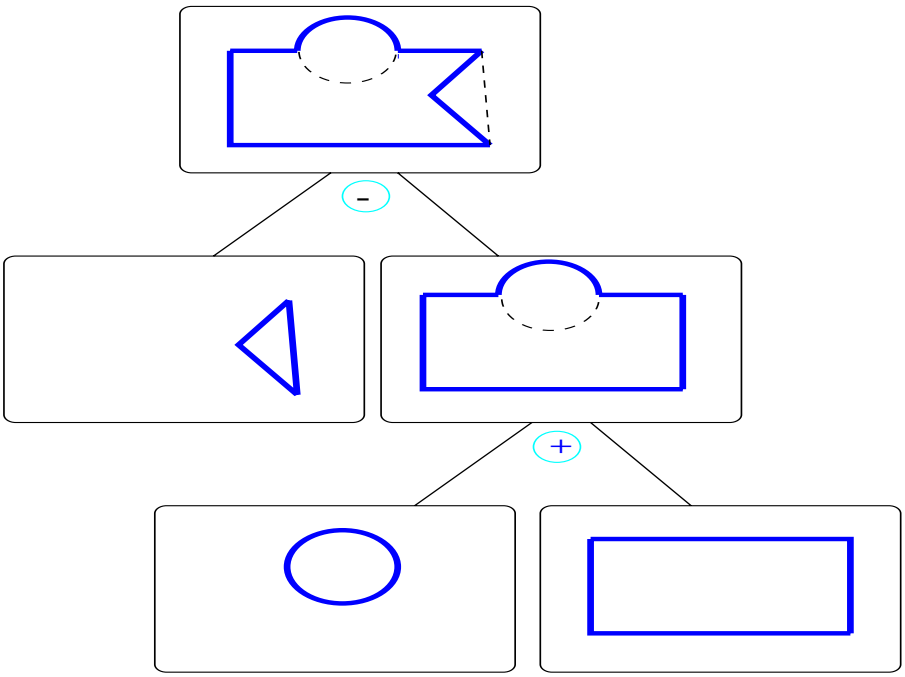

Figure 1. Logical operations as a binary tree representation.

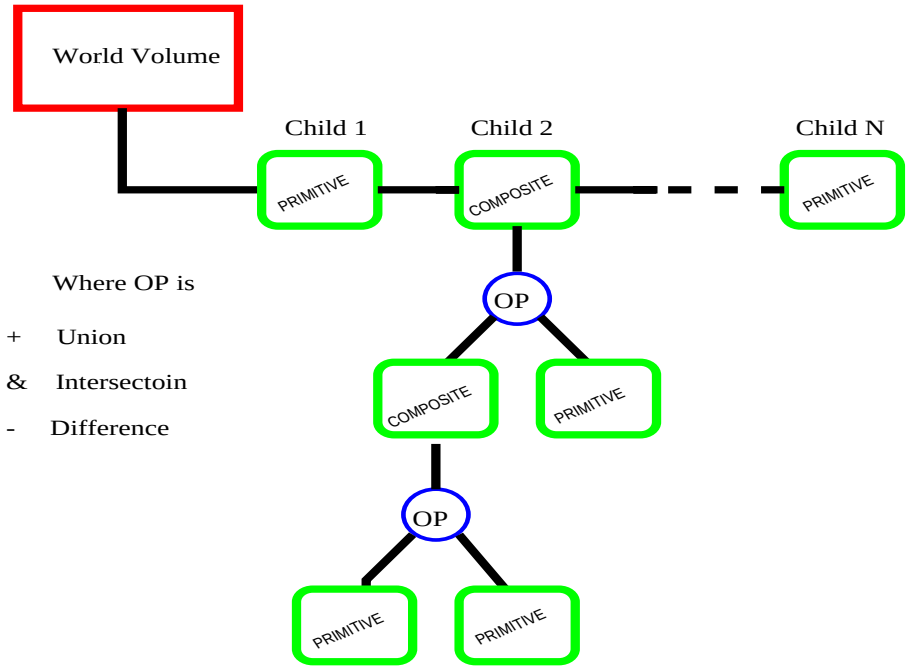

Figure 2. Memory layout of the geometry module.

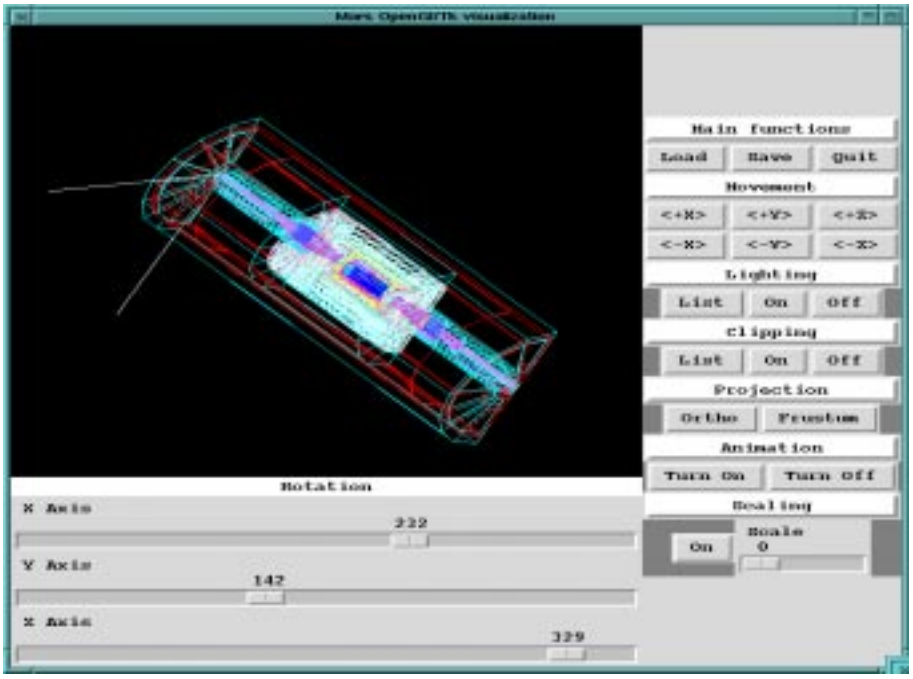

Figure 3. CMS detector, wire-frame representation.

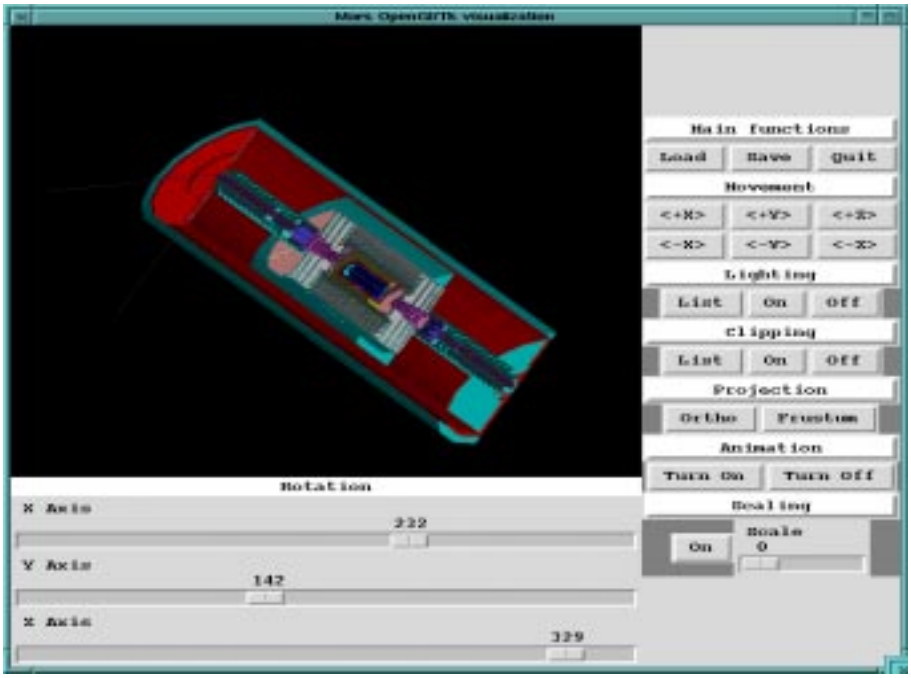

Figure 4. CMS detector, solid body representation.